Machine Learning W5. Neural Network Learning.

1. Cost Function.

- neural network for classification.
- L: total # of layers.
- s_l: # of units in layer l.

(1) Binary classification.

$y = 0$ or 1. ; 1 output unit. $h_\Theta(x) \in \mathbb{R}$.

$s_L = 1$, $K = 1$.

(2) Multi-class classification.

$y \in \mathbb{R}^K$. e.g. $\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$,

K output units.

$h_\Theta(x) \in \mathbb{R}^K$. $s_L = K$. $(K \geq 3)$

<u>Cost function</u>

Logistic regression. $J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))\right]$

$$+ \underbrace{\frac{\lambda}{m}\sum_{j=1}^{n} \theta_j^2}_{\text{regularization.}}$$

$\downarrow$ generalized.

Neural network. $h_\Theta(x) \in \mathbb{R}^K$; $(h_\Theta(x))_{(i)} = i^{th}$ output.

$$J_\Theta(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \cdot \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1 - h_\Theta(x^{(i)}))_k\right]$$

sum over output units

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} (\theta_{ji}^{(l)})^2$$

2. Back Propagation Algorithm

$\min\limits_{\Theta} J(\Theta)$ $\longrightarrow$ need to compute $J(\Theta)$

$$\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)}$$

(1) Gradient computation

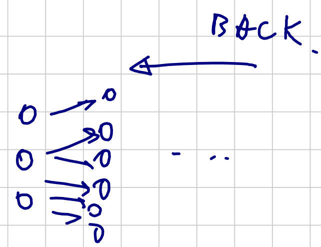
Consider only a single training example (x, y)

Forward propagation.

- $a^{(1)} = x$
  $z^{(2)} = \Theta^{(1)} a^{(1)}$
- $a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

⋮

- $a^{(4)} = h_\Theta(x) = g(z^{(4)})$



Back propagation

intuition. $\delta_j^{(l)}$ = "error" of node j in layer l.

recall: $a_j^{(l)}$ activation unit in ..

for each output unit. (layer $L=4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$ → "labelled" training set.

vectorized.

$$\delta^{(4)} = a^{(4)} - y \in \mathbb{R}^K$$ # of output unit.

back.

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* \boxed{g'(z^{(3)})}$$ → $a^{(3)} .* (1-a^{(3)})$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$ → $a^{(2)} .* (1-a^{(2)})$

($g = \frac{1}{1+e^{\theta^T x}}$.

It can be shown that:

$$\boxed{\begin{aligned}&\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \cdot \delta_i^{(l+1)} \\ &(\text{for } \lambda = 0)\end{aligned}}$$

$\Theta^{(l)}$ maps: $a^{(l)} \to a^{(l+1)}$

<u>Backpropagation Algorithm</u>

Training set $\{ (x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)}) \}$

Set $\Delta_{ij}^{(l)} = 0$ for all l, i, j ($\longrightarrow$ used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1:m$ %. loop thru. training set.

① set $a^{(1)} = x^{(i)}$ $\to$ input for ith training example.

② perform forward propagation to compute $a^{(l)}$ for $l = 1, 2, 3 \ldots L$

③ Using $y^{(i)}$, $\delta^{(L)} = a^{(L)} - y^{(i)}$

$\delta^{(L-1)}, \delta^{(L-2)} \ldots$

$$\Delta_{ij}^{l} := \Delta_{ij}^{l} + a_j^{(l)} \cdot \delta_i^{(l+1)}$$

$\downarrow$ vect.

$$\underset{(\text{matrix})}{\Delta^{l}} = \Delta^{l} + \delta^{(l+1)} (a^{(l)})^T$$

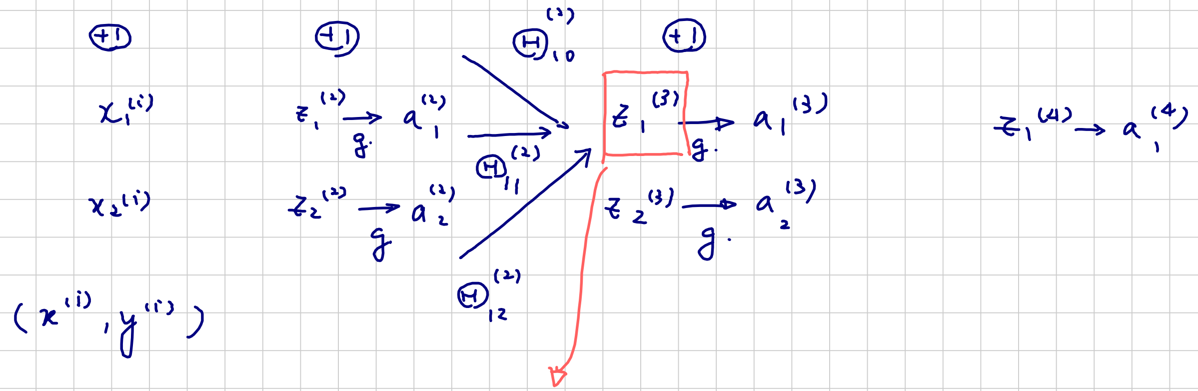
$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}} J(\Theta) = D_{ij}^{(l)}$$

3. Backpropagation Intuition.

(1) Forward propagation.



$$z_1^{(3)} = \Theta_{10}^{(2)} \cdot 1 + \Theta_{11}^{(2)} \cdot a_1^{(2)} + \Theta_{12}^{(2)} \cdot a_2^{(3)}$$

Now, notice that backpropagation is doing something very similar to this, but in the reversed dir. ($\leftarrow$) instead of ($\longrightarrow$)

(2) What is back propagation doing?

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log[h_\Theta(x^{(i)})] + (1-y^{(i)}) \cdot \log(1-h_\Theta(x^{(i)}))\right]$$

$$\underbrace{+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} [\Theta_{ji}^{(l)}]^2}$$

(for now, assume $\lambda = 0$, ignore regularization)

- focus on single example $(x^{(i)}, y^{(i)})$, case of 1 output unit.
- Consider cost: $\text{cost}^{(i)} = y^{(i)} \cdot \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \cdot \log(h_\Theta(x^{(i)}))$

  $\text{cost}^{(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$

  $\hookrightarrow$ how close is $y^{(i)}$ to output $h_\Theta(x^{(i)})$?

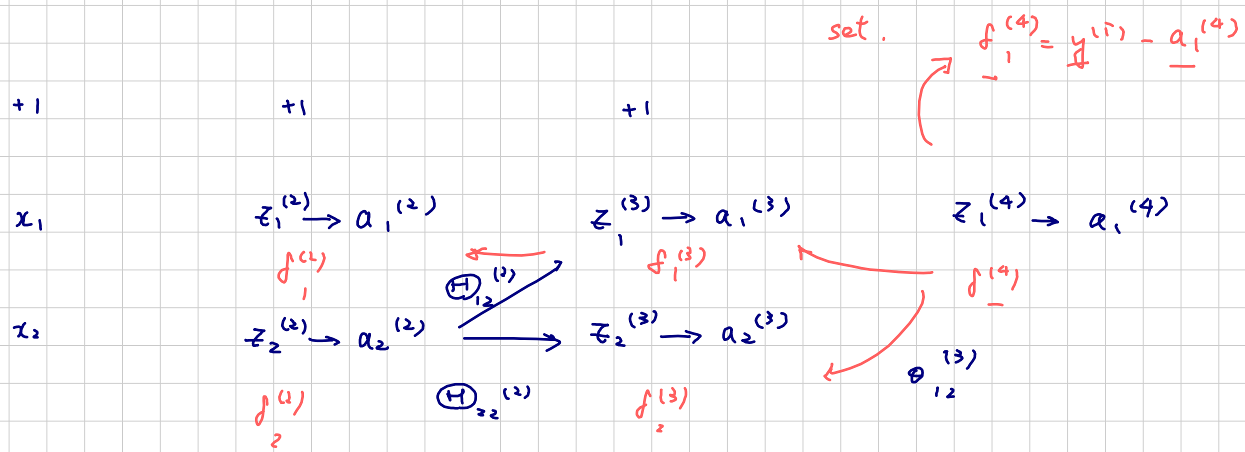
Now, $\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l)

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}^{(i)}$ for $j \geq 0$, where

$$\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) . \log h_\Theta x^{(i)}$$

↳ measure of how much we need to change $z_j^{(l)}$ to affect $h_\Theta(x^{(i)})$.

[partial derivates of cost func. wrt. intermediate values, so as to affect the output].



① How did we arrive at $\delta_2^{(2)}$? "Weighted sum of the activation unit in $l+1$ layer"

$$\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)}$$

② $\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$.

Machine Learning Week 5 Pt. 2 : Backpropagation in Practise.

4. Implementation Note: Unrolling Parameters.

Advanced optimization

function [jval, gradient] = costFunction (theta)

$\hookrightarrow \mathbb{R}^{n+1}$. $\hookrightarrow \mathbb{R}^{n+1}$ (vectors)

...

optTheta = fminunc (@costFunction, initialTheta, options)

Neural Network (L = 4) → not "vectors" anymore, instead "matrices"

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)

$D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D_1, D_2, D_3)

↳ Unroll into vectors.

· reshape → slice elements and fit into matrix.

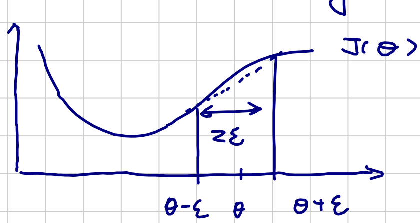
Learning Algorithm.

· Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ → unroll to get initialTheta.

5. Gradient Checking

· eliminates error from escaping verification. (BP is error-prone but sometimes not detectable)

· Numerical estimation of gradients.



- get secant line btwn (θ-ε, θ+ε)

$$\frac{dJ(\theta)}{d\theta} \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}, \quad \varepsilon \sim 10^{-4}.$$

$$\left(\text{alt.} \quad \frac{dJ(\theta)}{d\theta} = \frac{J(\varepsilon+\theta) - J(\theta)}{\varepsilon} \right)$$

When, $\theta \in \mathbb{R}^n$. [unrolled version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$]

so $\theta = (\theta_1, \theta_2, \dots \theta_n)$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \dots \theta_n) - J(\theta_1-\varepsilon, \theta_2, \dots \theta_n)}{2\varepsilon}$$

change only 1. and hold other. const.

$\vdots$

$$\frac{\partial}{\partial \theta_n} J(\theta)$$

$\to$ approxing $\nabla J(\theta)$

. key. Check $\boxed{\text{grad} \approx \text{DVec}}$ $\to$ from backprop.

<u>Note</u>

very expensive.

① BP to compute DVec. [unrolled $D^{(1)}, D^{(2)}, D^{(3)}$]

② Implement numerical grad. check. to compute gradApprox.

③ Make sure they give similar values.

[$D, \delta^{(1)}, \delta^{(2)} \dots$]

(*) ④ Turn off gradient checking, use backprop. for learning.

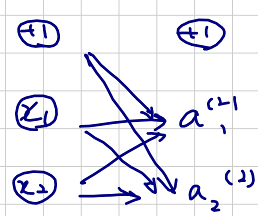
$\to$ Remember! Disable gradient checking else code will run slow

6. Random Initialization.

. We need initial value for Θ

-set initialTheta = zeros(n,1) $\to$ logistic ✓

- NN.



$\Theta_{ij}^{(l)} = 0 \quad \forall\, l, i, j$

After each update, weight pairs are still same (diff. value) btwn. each other.

$a_1^{(2)} = a_2^{(2)}$, also $\delta_1^{(2)} = \delta_2^{(2)}$

$$\Rightarrow \frac{\partial}{\partial \Theta_{01}^{(1)}} J(\theta) = \frac{\partial}{\partial \theta_{02}^{(1)}} J(\Theta) \Rightarrow \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$$

⇒ Random Initialization : symmetry breaking.

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

(i.e. $-\varepsilon \le \Theta_{ij}^{(l)} \le \varepsilon$)

not same ε in prev. sect.

7. Summary.

(1) Training a neural network.

- pick a network architecture (connectivity)

  - layers, # of act. units.

  input unit : dimension of features $x^{(i)}$.
  output " : # of classes.

multiclass → $y = \{1, 2, 3, \ldots, 10\}$. $y = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ or $\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

$[y=1]$ $[y=2]$.

- Reasonable default.

  1 hidd. layer

  ≥ 1 hidd layer, w/ same # of hidd. units.

  (the more the better) ✓ ⟺. complex ↑

1) Randomly initialize weights

2) FP : $h_\Theta(x^{(i)})$ $\forall x^{(i)}$

3) $J(\Theta)$

4) BP : $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

- for loop over examples (get $a^{(l)}$ and $\delta^{(l)}$ $l = 2, \cdots L$)

5) Gradient checking : comp. $\frac{\partial}{\partial \theta_{jk}^{(l)}} J(\Theta)$ BP

$\updownarrow$

num. est. of $\frac{\partial}{\partial \theta} J(\Theta)$

disable

6) $\left.\begin{array}{l}\text{Grad. descent} \\ \text{Adv. opt. method}\end{array}\right\}$ BP to min. $J(\Theta)$ over Θ.

- $J(\Theta)$ non-convex $\Rightarrow$ can get stuck in local minima.

↳ good enough in practise.